Embedded Linux Conference North America

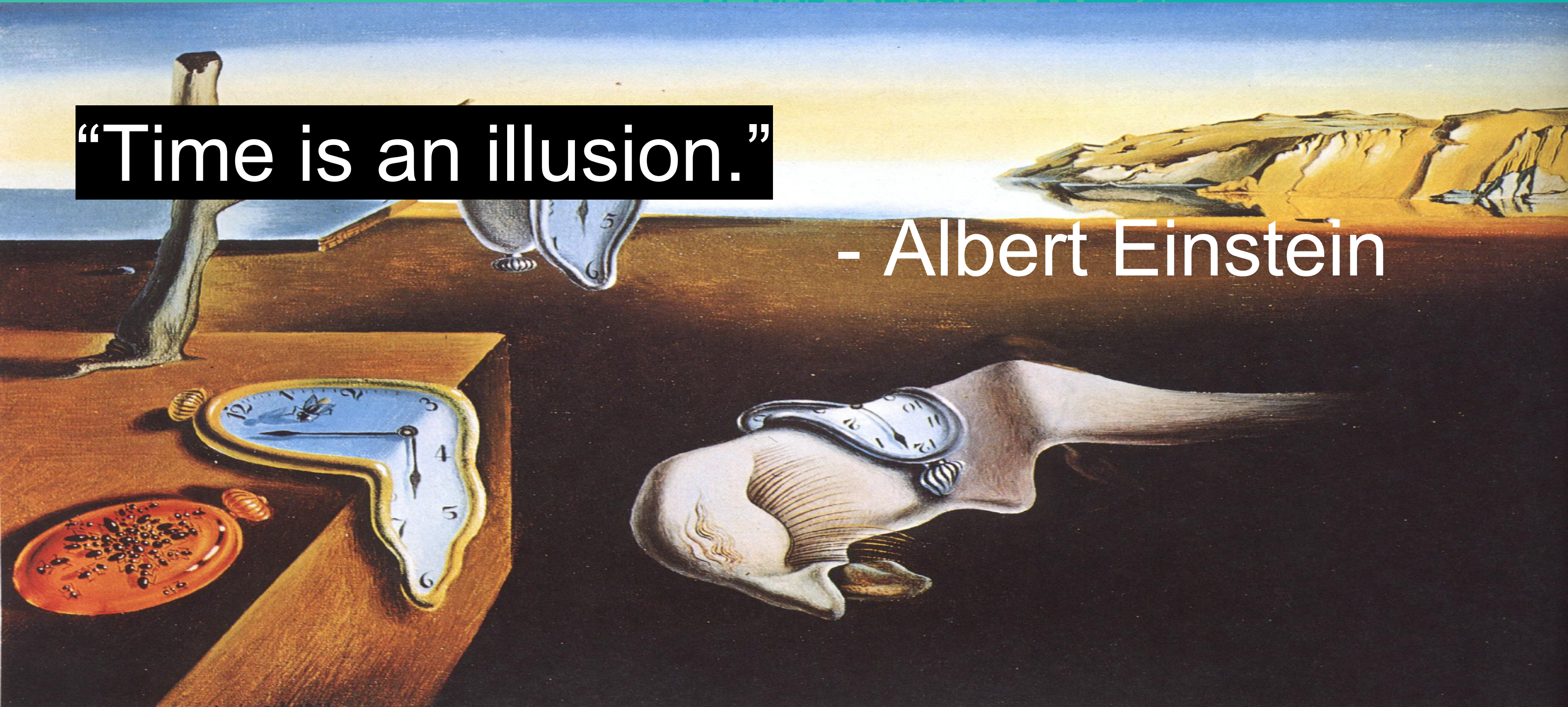
OpenIoT Summit North America

Not Really, But Kind of Real Time Linux

Sandra Capri, CTO, *Ambient Sensors, LLC*

sandra.capri@ambientsensors.com

THE LINUX FOUNDATION

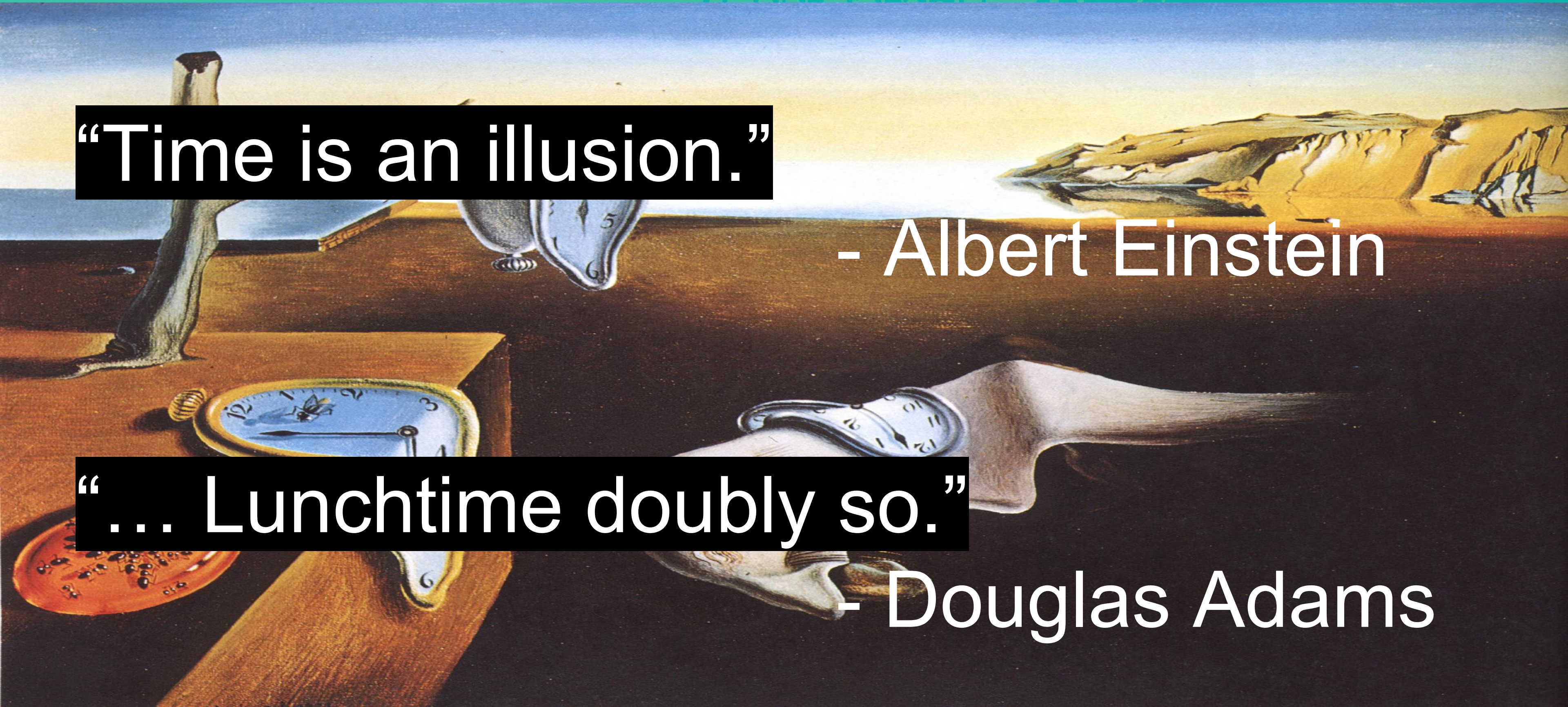


“Time is an illusion.”

- Albert Einstein

Embedded Linux Conference North America

OpenIoTSummit North America



“Time is an illusion.”

- Albert Einstein

“… Lunchtime doubly so.”

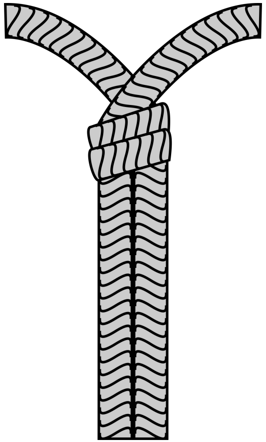
- Douglas Adams

Question

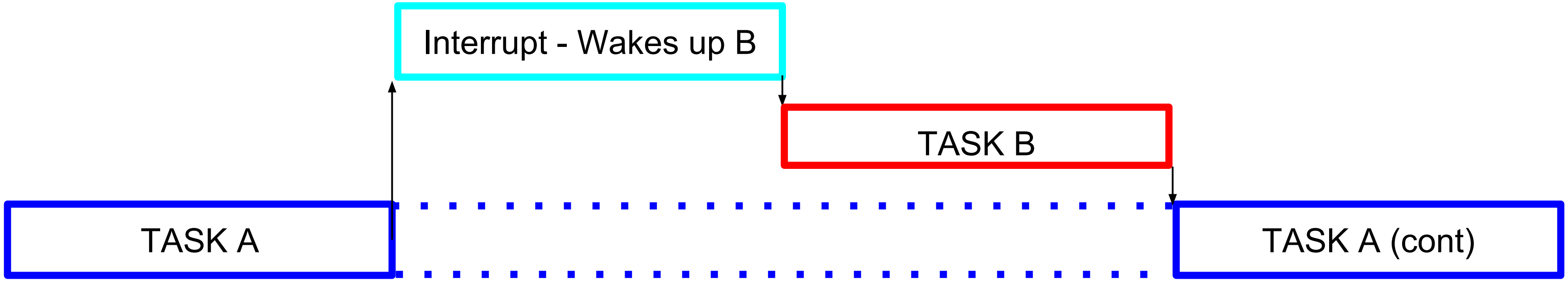
Question

Does a small embedded Linux platform have enough determinism to serve as a device controller?

Why would this be difficult for a Linux system to do anyway?



Why would this be difficult for a Linux system to do anyway?



Hard Real Time

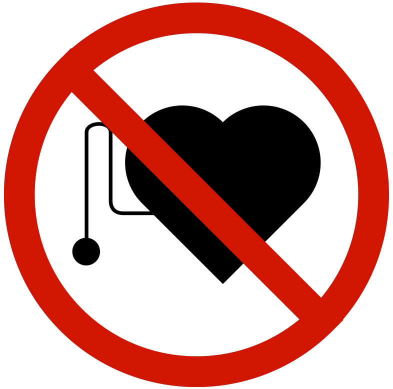
...any missed deadline is a system failure.

Soft Real Time

… allows for frequently missed deadlines, and as long as tasks are timely executed, their results continue to have value.

My Disclaimer

If it’s a matter of life or death, please don’t depend on soft real time, or anything I might say in this talk



Real Time Linux

PREEMPT_RT patches - removes all unbounded latencies

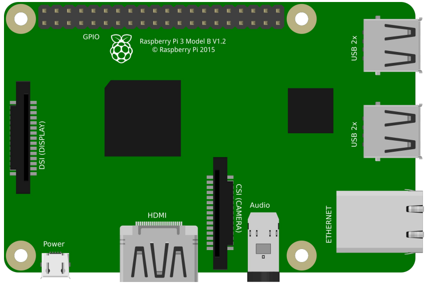
Real Time Linux Presentations

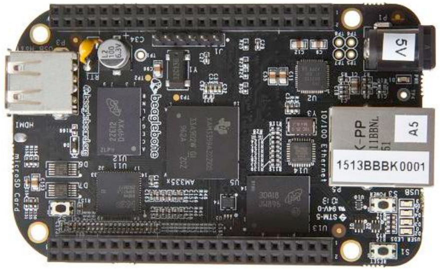
Andreas Ehmanns’ 2017 ELC talk -
“Real Time Linux on Embedded Multicore Processors”

Julia Cartwright’s 2018 ELC - “What Every Driver Developer Should Know About RT”

Can a Linux SBC control a GPIO to create a "very accurate" pulse width ?

Targets: Raspberry Pi 3 and Beaglebone Black.





Pulse Train - Assert and Deassert a GPIO to generate a series of pulses.

Measure to see if they meet the “very accurate” pulse width criteria.

What is the Most Accurate Method?

- Busy/wait loop (e.g. udelay)
- Allow the OS put the process to sleep?
- Kernel Space vs. User Space Accuracy

User-Space Program - first pass

1. Mark the time & assert the GPIO line.
2. Usleep (not a busy/wait).
3. Clear the GPIO line & mark the time.
4. Compare times and keep extremes

Results: no latency minimization (control)

```
ideal pulse   shortest pulse   longest pulse
(usec)         (rounded usec)   (rounded usec)
--------------------------------------------------------------------------
10,000          10,060             14,000 (some times > 20,000 usec)
 1,000           1,060              7,100 (some times > 9,000 usec)
   100             140              2,900 (some times > 6,200 usec)
    10              30              1,500 (some times > 6,900 usec)
```

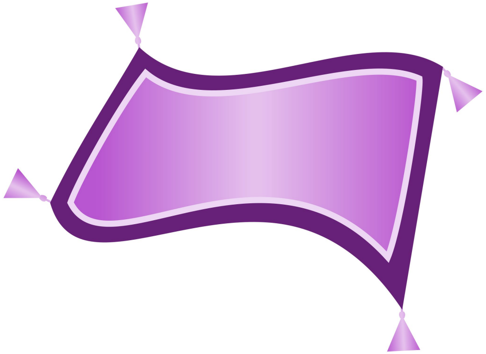
Wait: What's up with These Long Times?

```
ideal pulse   shortest pulse   longest pulse
(usec)         (rounded usec)   (rounded usec)
--------------------------------------------------------------------------
10,000         10,060             14,000 (some times > 20,000 usec)
 1,000          1,060              7,100 (some times > 9,000 usec)
   100            140              2,900 (some times > 6,200 usec)
    10             30              1,500 (some times > 6,900 usec)
```

Busy-work user space processes!

Ways to stop Linux from pulling the rug (core) out from under us

- Scheduling policy/priority
- Reserve a core
- Lock the task in memory

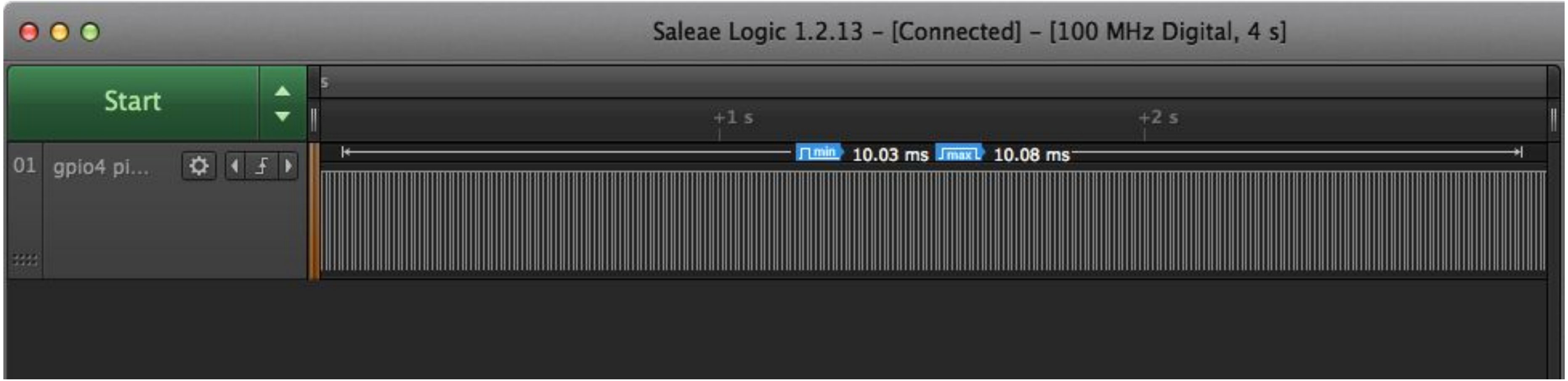


User Space Results (minimizing latency)

ideal pulse (usec)	shortest pulse (rounded usec)	longest pulse (rounded usec)
10,000	10,024	10,092
1,000	1,009	1,097
100	109	179
10	15	97

The times look much better with these system tweaks

User Space Results (minimizing latency)



Kernel Space

- Scheduling policy/priority
- Reserve a core
- Direct GPIO writes
- Disable kernel preemption
- Sleep vs. busy/wait
- CPU stalls & run time throttling

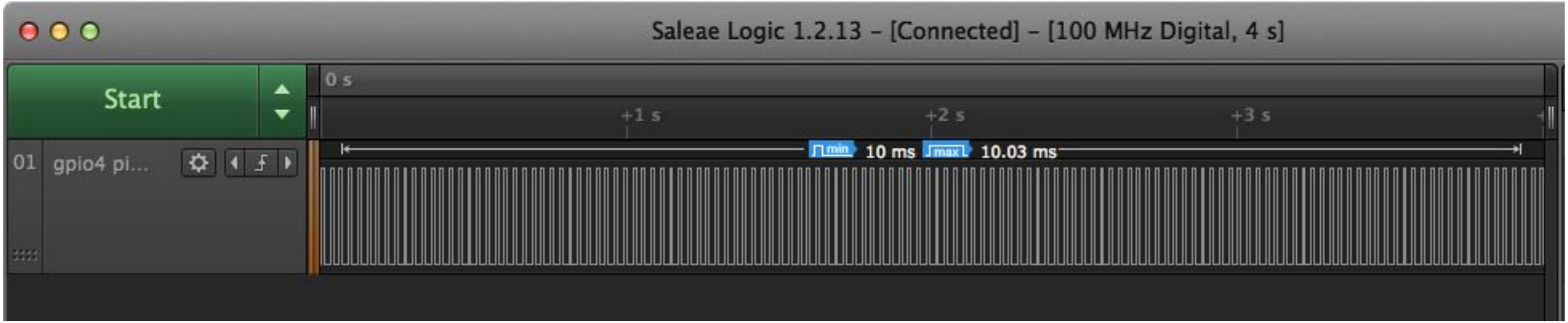
Kernel Results - Kernel Preemption On

ideal pulse (usec)	shortest pulse (rounded usec)	longest pulse (rounded usec)	average	wait mechanism
10,000	10,009	10,093	10,040	usleep (kernel sleep)
1,000	1,007	1,095	1,036	usleep (kernel sleep)
100	107	151	114	usleep (kernel sleep)
10	14	74	22	usleep (kernel sleep)

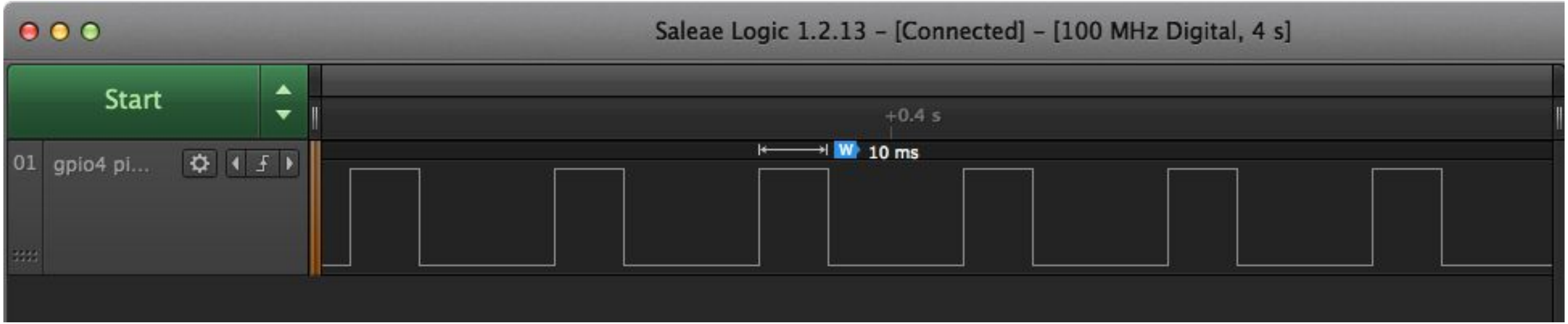
Kernel Results - Kernel Preemption On

ideal pulse (usec)	shortest pulse (rounded usec)	longest pulse (rounded usec)	average	wait mechanism
10,000	10,000	10,072	10,005	udelay (busy wait)
10,000	10,009	10,093	10,040	usleep (kernel sleep)
1,000	1,000	1,062	1,002	udelay (busy wait)
1,000	1,007	1,095	1,036	usleep (kernel sleep)
100	100	170	101	udelay (busy wait)
100	107	151	114	usleep (kernel sleep)
10	10	87	11	udelay (busy wait)
10	14	74	22	usleep (kernel sleep)

Kernel Results - Kernel Preemption On



Kernel Results - Kernel Preemption On



Kernel Results - Kernel Preemption Off

ideal pulse (usec)	shortest pulse (rounded usec)	longest pulse (rounded usec)	average	wait mechanism
10,000	10,000	10,063	10,003	udelay (busy wait)
1,000	1,000	1,060	1,002	udelay (busy wait)
100	100	152	100	udelay (busy wait)
10	10	88	11	udelay (busy wait)

Kernel - Do we really own the core?

```
top - 20:55:43 up 27 min,  6 users,  load average: 236.44, 536.81, 303.56
Tasks: 136 total,   3 running, 133 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us, 25.2 sy,  0.0 ni, 74.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:    947684 total,    75996 used,   871688 free,      196 buffers
KiB Swap:   102396 total,    41580 used,    60816 free.    11260 cached Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND	P
19	root	rt	0	0	0	0	S	0.0	0.0	0:00.00	migration/3	3
20	root	20	0	0	0	0	S	0.0	0.0	0:00.00	ksoftirqd/3	3
21	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kworker/3:0	3
22	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/3:0H	3
1066	**root**	**20**	**0**	**0**	**0**	**0**	**R**	**0.0**	**0.0**	**0:00.00**	**kworker/3:1**	**3**
16287	**root**	**rt**	**0**	**1912**	**0**	**0**	**R**	**100.0**	**0.0**	**3:57.66**	**sh**	**3**
1	root	20	0	23916	2448	1828	S	0.0	0.3	0:08.11	systemd	2
8	root	20	0	0	0	0	S	0.0	0.0	0:00.01	rcu_sched	2
15	root	rt	0	0	0	0	S	0.0	0.0	0:00.57	migration/2	2

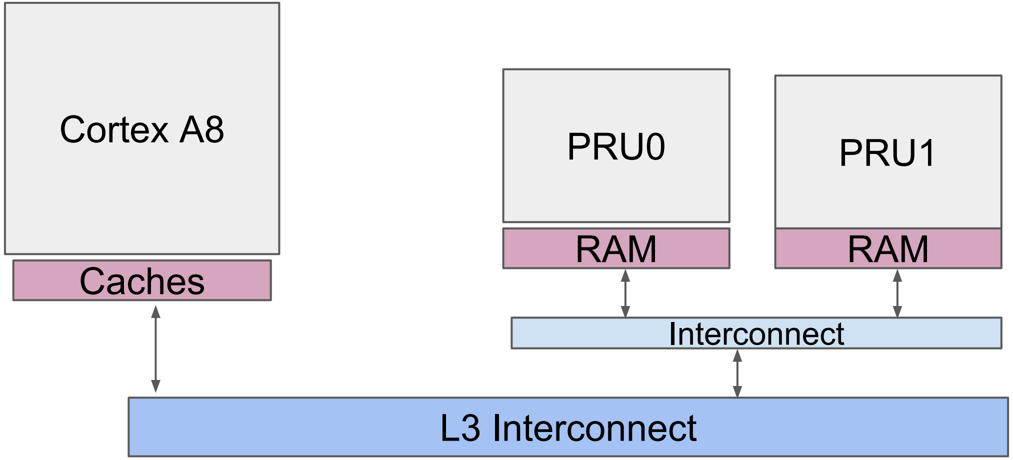
Kernel - Do we really own the core?

```
top - 17:28:23 up  1:20,  6 users,  load average: 639.78, 295.22, 184.27
Tasks: 2396 total,   8 running, 2382 sleeping,   2 stopped,   4 zombie
%Cpu(s): 14.5 us, 84.4 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  1.2 si,  0.0 st
KiB Mem:    947684 total,   491496 used,   456188 free,     3160 buffers
KiB Swap:   102396 total,    64528 used,    37868 free.    17044 cached Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND	P
19	root	rt	0	0	0	0	S	0.0	0.0	0:00.00	migration/3	3
20	root	20	0	0	0	0	S	0.0	0.0	0:00.00	ksoftirqd/3	3
21	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kworker/3:0	3
22	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/3:0H	3
10147	**root**	**rt**	**0**	**1912**	**392**	**332**	**R**	**100.0**	**0.0**	**1:42.17**	**sh**	**3**
14279	**root**	**20**	**0**	**0**	**0**	**0**	**R**	**0.0**	**0.0**	**0:00.00**	**kworker/3:1**	**3**
1	root	20	0	22860	2880	2312	S	0.0	0.3	0:06.21	systemd	2
7	root	20	0	0	0	0	S	0.3	0.0	0:01.40	rcu_preempt	2
15	root	rt	0	0	0	0	S	0.3	0.0	0:00.39	migration/2	2

BeagleBone Black: Just a Single Core?

- Single core ARM (for Linux)
- 2 PRU cores



Linux and the PRUs

- No Linux on the PRUs; write your own thread.
- No OS latency!
- The PRUs and the A8 can communicate

See Rob Birkett's 2015 ELC presentation:
"Enhancing Real-Time Capabilities with the PRU".

How to use the PRUs

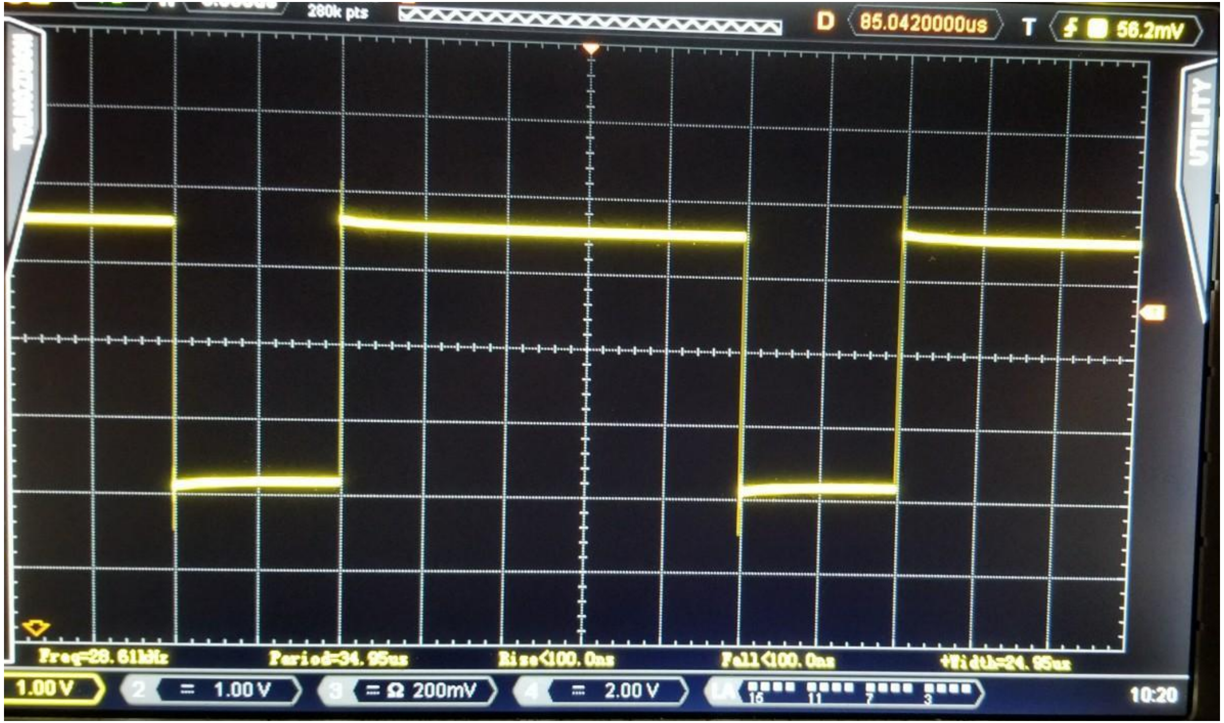
- Write firmware for the PRUs
- Use Linux driver to load it on the PRUs
- Configure with DT entries
- Set up comm between A8 and PRUs

See Jason Kridner's video: "Using the BeagleBone Real-time Microcontrollers" at http://beaglebone.org/pru

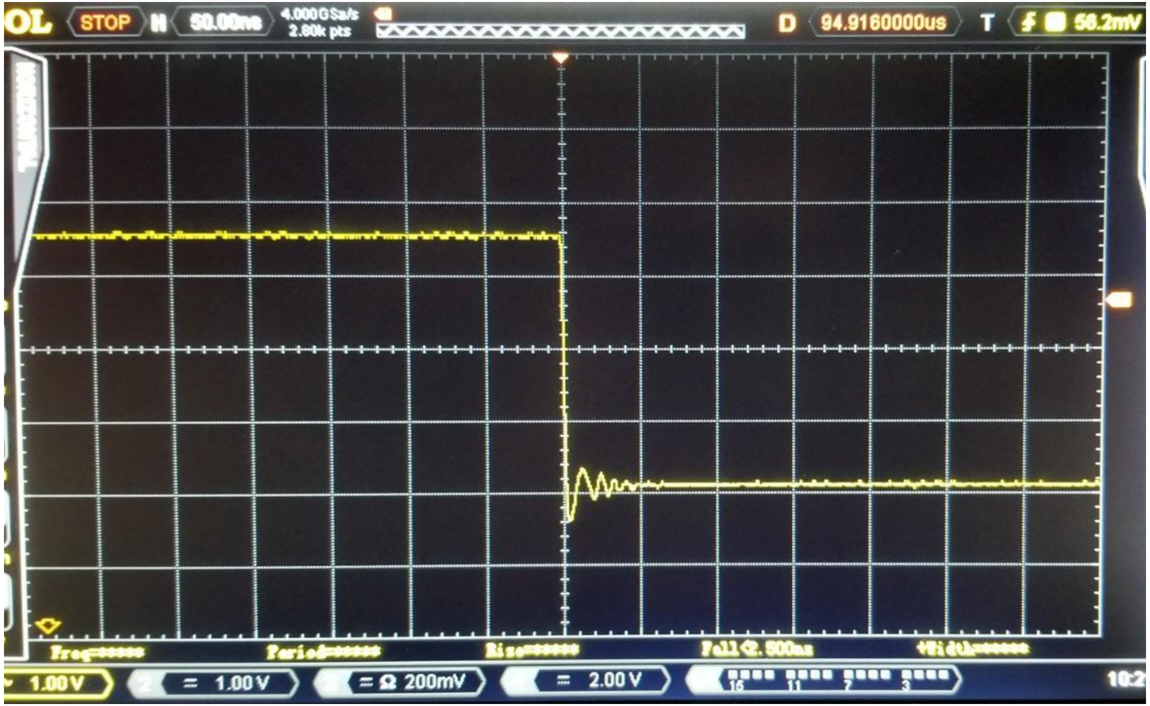
BeagleBone Black Results

- Woah - PRUs are very accurate.
- Error on order of 10s of nanoseconds.
- Forget the 10msec pulse, how does a 25usec pulse look?

BeagleBone Black 25 usec pulses



BeagleBone Black 25 usec pulses

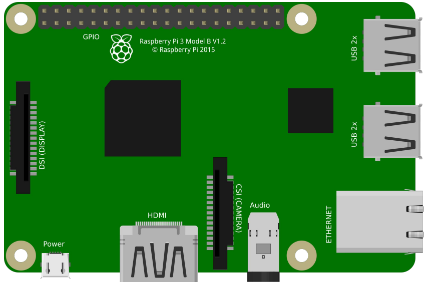


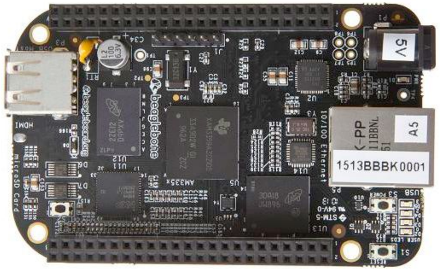
What About The Busy Work Processes?

- No impact on the PRU: they run on the A8 core.
- Heavily-loaded system should not affect the PRU

So Which One to Use? BBB or RPi?

- Do you need deterministic control, and feel comfortable writing threads for the PRUs?
- Do you prefer to keep your code in Linux, and are ok with delays in the ~100usec range?





Possible Future Investigation

- IRQs - max delay from event to ISR running (RPi: interrupt affinity, BBB: dedicated PRU)
- Cache misses (lock code in L1 cache)
- Investigate advantages to driving SPI instead of GPIO?

To get to the Whitepaper

https://www.ambientsensors.com

Click on the “Downloads” tab (which is currently under the “Game Changers” tab)

sandra.capri@ambientsensors.com

Embedded Linux Conference
North America

OpenIoT Summit
North America

THE LINUX FOUNDATION